Experimental Learning Event at Legna Software

Analyzing Oberlin College's Environmental Curriculum as a Successful Network

A Thesis Prospectus
In STS 4500
Presented to
The Faculty of the
School of Engineering and Applied Science
University of Virginia
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By
Matthew Gerace

December 9, 2022

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISORS

Benjamin Laugelli, Department of Engineering and Society

Brianna Morrison, Department of Computer Science

Introduction

Students transition to software engineers biannually after completing the University of Virginia's computer science curriculum. Software engineering is "a detailed study of engineering to the design, development, and maintenance of software," yet the University of Virginia teaches computer science, "the study of computers and computational systems" ("What is 'software engineering'?," n.d.; *What is computer science?*, n.d.). There is a critical disparity between engineering and science, especially for software. Computer knowledge does not directly translate to software development ability, much like biology knowledge does not directly translate to surgical skills. The computer science curriculum, as it stands, has limitations for producing future competent software engineers. When the University of Virginia educates computer science students for software engineering careers, they shape the future of technology. The engineers they develop and their contributions advance science, technology, and society. The resulting competition for distinguished program enrollment cultivates software innovation and, thus, many of the world's operations.

I was unprepared as a software engineer during an experimental learning event at Legna Software. My courses did not thoroughly develop my ability to solve open-ended problems with end-to-end software solutions. To combat this unpreparedness, I will propose changes to address limitations in the computer science curriculum that will better prepare future software engineers. These adjustments will effectively shape technology's future by changing today's students.

Education reform is a complex process. Human and non-human factors and their relationships in this system determine success. It is not enough to introduce new learning technology; successful social interactions between all contributors are necessary for education change. This idea requires that I not only outline specific technical adjustments for curriculum

reform but also examine the human and non-human relationships that invoke success. I will specifically analyze Oberlin College's 2004 environmental policy, which successfully influenced students' post-graduate environmental contributions to understand these relationships better.

Social relationships and technical changes are both necessary to adjust the computer science curriculum. I will discuss the aspects of my internship experience that left me unprepared and propose adjustments to prepare students with similar experiences better. I will also leverage actor-network theory over Oberlin College's 2004 education reform to examine the relationship between human and non-human actors. Understanding how these connections influence curriculum revisions that successfully improve associated students' societal impact is critical for reshaping the computer science curriculum.

Technical Project Proposal

I interned with Legna Software in the Summer of 2022. Legna Software is a technology startup located in Winston-Salem. Its objective is to digitalize the forestry industry by leveraging a software-as-a-service model to provide wood mills with real-time financial and logistical information. One particular challenge is auditing mill-to-mill transactions. These transactions result from obtaining excess or unnecessary trees, limbs, or branches while acquiring lumber. Mills then sell their unwanted lumber instead of wasting resources to procure only necessary materials. Through this process, the seller only receives information after the lumber is obtained and delivered and the driver returns. As a result, any issues regarding specifications or pricing come to fruition after two to four weeks. In response, I implemented an end-to-end software application to make mill-to-mill receipt information instantly available on Legna Software's software-as-a-service model.

For this endeavor, I leveraged many computer science fundamentals, such as algorithms, data structures, cloud computing, and machine learning. Likewise, I used techniques I learned from these specific courses. I predominantly tapped into the software development principles introduced in the course Advanced Software Development. This class, other non-computer science courses, and extracurricular activities prepared necessary soft skills. Leadership, responsibility, collaboration, persistence, and curiosity were essential as I worked towards constructing an end-to-end application from the ground up.

While I had building blocks through academic, extracurricular, and personal experiences, the project differed significantly from these past experiences in two areas. These two areas were the specific tools and technologies required and the methodological differences between development production and academia. In my classes, I relied on programming languages such as C++, Python, and Java, cloud providers such as Google Cloud Platform and Amazon Web Services, and SQL databases. However, my project required C# for my programming language, Microsoft Azure for my cloud provider, and a Non-SQL database– all of which I had previously never utilized. Additionally, I mainly developed in iterations for an academic setting, but agile development was a strict requirement for a production setting. Overall, my education needed a stronger focus on modern tools, software development practices, and building end-to-end applications.

My education prepared me to learn necessary information quickly; nevertheless, its neglect of practical software engineering limited my experience's potential. Given a change in the computer science curriculum, I would have had more opportunities to learn about the products, infrastructure, operation, and details important for working at a startup. Realizing these opportunities would have strengthened my software engineering ability and early career path.

The computer science curriculum in its current form does not fully prepare students for an experiential learning event like mine. Implementing proven commercial software course practices, the most modern technology, and real-world development opportunities, such as open source contribution, within the entire curriculum would better prepare students.

To instill these outlined changes, I will draw upon successful commercial software course methods. These include “having polished set of lectures with many real-life anecdotes” and “applying the lecture material to a real product” to “support[ing] transfer from classroom examples to workplace application” (Kaner & Fielder, 2005). Introducing open-sourced contributions in the classroom will provide students with opportunities for real-world production with modern technology. Open source is publicly accessible code that anyone can modify. In education, open source “not only provides [students] with a world-size laboratory and support staff but also gives them experience in large-scale software collaboration and development” (O'Hara & Kay, 2003). I will formulate these concepts by referencing courses that were influential to my experience, such as the syllabus for Advanced Software Development–a course that “analyzes modern software engineering practice” (Sherriff & McBurney, 2022). I will also reference proven adjusted classroom structures documented in “Inside out: A computer science course gets a makeover” and analyze “Open source software and computer science education” to lay out a commitment to real-world software contributions in the curriculum. Together these concepts derived from these sources will formulate computer science curriculum revisions that better prepare students for an experience like mine.

STS Project Proposal

The Oberlin College (Oberlin) board of trustees approved a new environmental policy in March 2004 (Gulasy, 2016). This policy serves as a strict commitment for developing students to address sustainability. The proposed ideas rivaled the previous curriculum in that Oberlin did not want their students to be solely aware of environmental damage and sustainable practices. Instead, they hoped their community would also apply this knowledge and create tangible solutions to enact real change. This transition was proposed "through teaching and research [and] through design and implementation of institutional policies" as outlined in *Oberlin College's environmental policy* (Baumann et al., 2004). This policy successfully sprouted a shift in the entire Oberlin community as eighteen years later, in 2022, Oberlin currently states that sustainability is a "defining factor for campus operations." (*Oberlin's climate commitment*, 2021). The policy's ability to break the traditional siloed structure for a sustainability curriculum and replace it with a university-wide involvement towards executing environmental change primarily influenced its success.

Greening the college curriculum: A guide to environmental teaching in the liberal arts (*Greening the college curriculum*) provides advice for adjusting curriculums, so students institute environmental change after graduation. This text cites Oberlin as an example of employing their practices. It broadly attributes "institutional resource flows," "developing data," and "environmental ideas and projects" (Collett, J. (Ed.), 1996, p. 21) as factors that were primarily responsible for Oberlin's curriculum's effective transition. These outlined entities contributed to this project's success, but this view overlooks other essential elements. Mainly, The authors of *Greening the college curriculum* neglect the relationship between these non-human and human factors and the social connections they cultivate. For example, the

institutional resource flows established a community-wide commitment to enacting environmental change rather than a strict student or professor classroom commitment. Furthermore, The data development and projects did not just help solve environmental challenges but also inspired communities fostering care and passion for tackling these problems.

If we only attribute the project's success to concrete actors, such as opportunities established by the curriculum, then we will not have a comprehensive account of the range of factors, including social relationships outside of the classroom, that contributed to Oberlin's curriculum success. Drawing on actor-network theory, I will argue that together curriculum changes and resulting positive relationships across all associated entities drove Oberlin's success in cultivating a revised environmental curriculum.

Actor-network theory is a framework that analyzes social relationships between heterogeneous human and non-human entities and how they contribute to the success or failure of a given system. This framework relies on the fundamental principle that no single actor within a network has greater autonomy in the system's results than another actor. While all actors influence success equally, each network has a network builder, typically a scientist or engineer. This unique network builder enacts translation by defining goals and methods, recruiting technical and non-technical actors, and assigning actors objectives for success (Callon, 1986, p. 26). Through these actions, it is crucial to distinguish network builders are world builders in which they are "attempting to build a world where bits and pieces, social, natural, physical, or economic, are interrelated [in a network]" (Law, 1987, p. 231). All actors must share a successful association for a network to succeed. Conversely, actor oversight subjects a network to fail.

To support my argument, I will analyze evidence from *Greening the college curriculum* and resources directly from Oberlin, including policies, blog posts, and current measures of

sustainability success for students. *Greening the college curriculum* provides direct information about the necessary steps to develop students to enact real environmental change after graduation through education structure and community. Additionally, it examines the implementation of their practices at Oberlin and what factors influenced success. Oberlin's documentation will provide direct information into the factors and relationships that contributed to their success in their revised environmental curriculum. These primary sources will allow me to argue that a combination of the technical factors highlighted in *Greening the college curriculum*, as well as actor relationships, influenced the success of this network.

Conclusion

My proposed changes to the computer science curriculum will deliver courses that better prepare students to engage in software engineering experiences requiring them to build innovative end-to-end software solutions like mine at Legna Software. Understanding the social relationships that contributed to the success of Oberlin's environmental curriculum with actor-network theory will inspire the social interactions my curriculum overhaul aims to foster. With these insights from my STS research, I hope to apply similar policies that shape a community-wide devotion to developing competent software engineers of the future. Specifically, I plan to cultivate community engagement, passion, and commitment to building world-altering software that binds the associations between all actors in my network. My proposed changes to the University of Virginia Education curriculum, coupled with the social relationships I hope to foster, will enable the University of Virginia and its software engineers to advance science, technology, and society, through expertise in software development.

References

Baumann, F., Benzing, D., Craig, N., Evans, A., Filardi, S., Filardi, R., Gaudin, S., Gerber, C., Jahns, C., Morgenstern, R., Orr, D., Person, J., Schildkraut, D,. Skinner, B., Turner, C. (May 2004). *Oberlin College environmental policy*. Retrieved from https://www.oberlin.edu/sites/default/files/content/office/environmental-sustainability/documents/environmental_policy.pdf

Callon, M. (1986). The sociology of an actor-network: the case of the electric vehicle. In Callon, M., Law, J., Rip, A. (Eds.), *Mapping the Dynamics of Science and Technology* (pg. 19-34). Palgrave Macmillan, London.

Collett, J. (Ed.). (1996). *Greening the college curriculum: A guide to environmental teaching in the liberal arts*. Island Press.

Gulasy, L. (2016, March 1). *General faculty adopts environmental policy implementation plan.* Oberlin College and Conservatory. Retrieved from https://www.oberlin.edu/news/general-faculty-adopts-environmental-policy-implementation-plan

Kaner, C., & Fielder, R. L. (2005). Inside out: A computer science course gets a makeover. In *Proceedings of The National Convention of the Association for Educational Communications and Technology 2*(1), 254-264.

Law, J. (1987, April 1). On the social explanation of technical change: The case of the Portuguese maritime expansion. *Technology and Culture, 28*(2), 227-252. Retrieved from https://doi.org/10.2307/3105566

Oberlin's climate commitment. (2021, April 22). Oberlin College and Conservatory. Retrieved from https://www.oberlin.edu/about-oberlin/mission-and-values/climate-commitment

O'Hara, K. J., & Kay, J. S. (2003). Open source software and computer science education. *Journal of Computing Sciences in Colleges*, *18*(3), 1-7.

Sherriff, M., & McBurney, W. (2022). *Course syllabus*. CS 3240. Retrieved from https://f22.cs3240.org/syllabus.html

What is computer science? (n.d.). Undergraduate computer science at the University of Maryland. Retrieved from https://undergrad.cs.umd.edu/what-computer-science

What is 'software engineering'? (n.d.). *The Economic Times*. Retrieved from https://economictimes.indiatimes.com/definition/software-engineering